CLAIMS

1. A computing device comprising a scheduler incorporating an algorithm for ordering the running of threads of execution having different priorities; and including a list of threads which are scheduled to run on the device, ordered by priority; the device further comprising at least one locking mechanism for blocking access to a resource of the device from all threads except for a thread that holds the locking mechanism; and in which a scheduled thread which is blocked from running causes the thread which holds the locking mechanism to run.

2. A computing device according to claim 1 wherein states are assigned to threads and the list comprises of all threads having a common state.

3. A computing device according to claim 2 wherein a blocked thread is not permitted to change its state.

4. A computing device according to any one of claims 1 to 3 wherein the list is subdivided in accordance with the priority of the threads it contains.

5. A computing device according to any one of the preceding claims wherein a thread is arranged to contain a pointer to any locking mechanism it is blocked on.

6. A computing device according to any one of the preceding claims comprising a plurality of non-nestable locking mechanisms.

7. A computing device according to any one of the preceding claims wherein the scheduler is arranged to be called at the end of an interrupt service routine which is caused to run on the computing device.

8. A computing device according to any one of the preceding claims wherein the locking mechanism(s) comprise(s) a mutex including a pointer, which is null if the mutex is free or points to the thread holding the mutex, and includes a flag indicating whether or not the mutex is contested.

9. A computing device according to claim 8 wherein the algorithm is arranged to delegate memory management to a replaceable memory model configured in dependence upon the configuration of the computing device.

10. A computing device according to claim 9 wherein the memory model is arranged to run in either pre-emptible or non-preemptible modes.

11. A computing device according to claim 10 wherein a mutex is arranged to protect the module from running in the pre-emptible mode.

12. A computing device according to any one of the preceding claims wherein the scheduler is included in a kernel of an operating system of the computing device.

13. A computing device according to claim 12 wherein the kernel comprises a microkernel or a nanokernel and where the threads are, respectively, microkernel or nanokernel threads.

14. A computing device according to claim 12 or 13 wherein the scheduler is arranged to be called each time the kernel is unlocked.

15. A computing device according to any one of the preceding claims comprising a mobile computing device.

16. A computing device according to claim 15 comprising a smart phone.

17. A method of operating a computing device, the method comprising providing a scheduler incorporating an algorithm for ordering the running of threads of execution having different priorities, and including a list of threads which are scheduled to run on the device, ordered by priority; providing at least one locking mechanism for blocking access to a resource of the device from all threads except for a thread that holds the locking mechanism; and arranging for a scheduled thread which is blocked from running to cause the thread which holds the locking mechanism to run.

18. An operating system for a computing device, the operating system comprising a scheduler incorporating an algorithm for ordering the running of threads of execution having different priorities, and including means for providing a list of threads which are scheduled to run on the device, ordered by priority; at least one locking mechanism for blocking access to a resource of the device from all threads except for a thread that holds the locking mechanism; and means for arranging for a scheduled thread which is blocked from running because the resource it requires is locked to cause the thread which holds the locking mechanism to run.

19. An operating system according to claim 18 wherein states are assigned to threads and the list comprises of all threads having a common state.

20. An operating system according to claim 19 arranged to inhibit a blocked thread from changing its state.

21. An operating system according to any one of claims 18 to 20 arranged to subdivide the list in accordance with the priority of the threads it contains.

22. An operating system according to any one of claims 18 to 21 wherein a thread is arranged to contain a pointer to any locking mechanism it is blocked on.

23. An operating system according to any one claims 18 to 22 comprising a plurality of non-nestable locking mechanisms.

24. An operating system according to any one of claims 18 to 23 wherein the scheduler is arranged to be called at the end of an interrupt service routine which is caused to run on the computing device.

25. An operating system according to any one of claims 18 to 24 wherein the locking mechanism(s) comprise(s) a mutex including a pointer, which is null if the mutex is free or points to the thread holding the mutex, and includes a flag indicating whether or not the mutex is contested.

26. An operating system according to claim 25 wherein the algorithm is arranged to delegate memory management to a replaceable memory model configured in dependence upon the configuration of the computing device.

27. An operating system according to claim 26 wherein the memory model is arranged to run in either pre-emptible or non-preemptible modes.

28. An operating system according to claim 27 wherein a mutex is arranged to protect the module from running in the pre-emptible mode.

29. An operating system according to any one of claims 18 to 28 wherein the scheduler is included in the kernel.

30. An operating system according to claim 29 wherein the kernel comprises a microkernel or a nanokernel and where the threads are, respectively, microkernel or nanokernel threads.

31. An operating system according to claim 29 or 30 wherein the scheduler is arranged to be called each time the kernel is unlocked.